Taylor J. Smith

Teaching Portfolio

Current as of
December 30, 2020

1 Biographical Sketch

I am a PhD candidate and teaching assistant in the School of Computing at Queen's University. Generally, I teach and assist with courses in theoretical computer science covering subjects such as formal language theory, automata theory, complexity theory, data structures, and algorithm analysis/design. I have also taught and assisted with courses on discrete and abstract mathematics.

I have considerable experience with delivering and managing undergraduate courses in all of the above topics, and I am interested in teaching undergraduate courses in these and related subjects, such as formal logic or cryptography. I am eager to develop advanced-level courses covering special topics in formal language theory, automata theory, and complexity theory, as well as independent study/reading courses in the same areas.

My teaching abilities have been recognized both by students and by the broader university community. I have received consistently high teaching evaluation scores, including the highest "effectiveness rating" of any instructor in the School of Computing at Queen's University in the Winter 2019 term. I have also received multiple teaching awards, including the university-wide Teaching Assistant/Teaching Fellow Excellence Award given to one graduate student per year at Queen's University.

2 Teaching Philosophy

> ...the best theory is inspired by practice and the best practice is inspired by theory.
>
> — Donald Knuth
> *Theory and Practice* (1991)

Teaching theoretical computer science presents a unique challenge: typically, at the level of instruction when most students are introduced to theory, all of their previous courses have been application-focused and, hence, directly applicable to what they expect to see when they enter the workplace. In a theoretical computer science course, then, where the concepts taught cannot be immediately connected to any real-world application, it is not unusual to hear the question "why do we care about this?" Indeed, having taught and assisted with courses that span the spectrum of theoretical computer science—algorithms, computability and complexity, data structures, discrete mathematics, formal languages and automata—this one question has been a universal constant. Students want to know why they are investing their time and energy into learning something that is too often presented in the abstract.

My attempt to guide students toward finding an answer to this question is best summarized by the above quote by Donald Knuth: I use practice to inspire theory and, in turn, I strive to have students' future practices inspired by this theory. As hard as some theorists may work to keep their field "pure", it is an undeniable truth that theory and practice are inextricably intertwined components of computer science. Rather than resisting this association, I propose that we embrace it: teach students from the very beginning about the deep and meaningful connections between theoretical computer science and other areas, so that they have a better understanding not only of theory, but of the field as a whole. My goal as an educator is to bring meaning to theory, and I make progress toward that goal by drawing three connections: between theory and application, between theory and the bigger picture, and between students and their education.

Connecting Theory to Application. One great pitfall of testing a student's understanding of a theoretical subject lies in asking "so what?" questions. These questions are often designed to test knowledge of definitions or theorems; for instance, "Show that the function $42n^8 + 20n^6$ is $O(n^{10})$". Answering questions in this vein provides an extrinsic motivation for, say, students preparing to write an exam. However, there is no intrinsic motivation for those same students to care about the solution they obtain.

I aim to design problems that require students to apply concepts taught in the classroom to a situation found in the real world. Often, I select applications I have not discussed in the classroom, leaving students to discover the link on their own and solidifying in their minds both the idea itself and how they might use the idea.

In terms of course design, I tend to allocate a higher percentage of marks to assignments and projects rather than to time-constrained assessments such as exams. I believe exams have their merits, but the fact that most exams are held at a point in the term when students no longer have an opportunity to improve their understanding severely limits the use of an exam as a learning tool. Placing a greater weight on assignments and projects encourages students to put more work into crafting a submission they can be proud of while, in the process, gaining a better understanding of how the material they learned can be applied to novel problems.

Connecting Theory to the Bigger Picture. When I teach concepts in theoretical computer science, I make an effort to connect the concept to its broader context. I often do so by outlining the history and development of the concept, or by highlighting a few areas of application where this concept may appear. For example, when I introduce students to Greibach normal form, I do not restrict the lesson simply to talking about context-free grammars. Rather, I introduce students to Sheila Greibach, how she developed her idea, and why compiler designers care about transforming grammars into the form bearing her name.

I include this broader context within my lectures not only to pique interest in otherwise dry topics, but also to imbue within students the sense that these topics exist outside of the realm of mathematics. Framing the introduction of a lesson through the lens of the broader arts and sciences—through biography, history, chemistry, or otherwise—both provides for a gentler introduction to an abstract concept and allows students to emerge from a course as more well-rounded academics.

Providing a historical perspective within lectures also comes with the added benefit of introducing students to the people behind the ideas, thus revealing that these ideas were not simply gifted unto us centuries ago by brilliant minds from atop the ivory tower. I feel that this approach encourages students to attempt their own discoveries, as the barrier to entry vanishes once students realize that the people who developed the very ideas being taught today were once just like them. This approach was influenced by texts such as Rosen's *Discrete Mathematics and Its Applications*, which includes biographical sidebars of mathematical figures.

Connecting Students to Their Education. When I was an undergraduate student, the theoretical computer science course in which I was enrolled was taught through PowerPoint. While this may work for some subjects, I staunchly believe that PowerPoint (and, indeed, any static presentation format) has no place in theoretical computer science. At its core, theoretical computer science is applied mathematics, and so much like a course in mathematics, instructors should employ a fluid, discovery-based approach to teaching theory.

To promote interest in a course, I invite students to participate in the development and presentation of course material during each lecture, rather than flicking through slides and beaming contextless information at the class. I write lecture material on the chalkboard, giving me the ability to amend and add to my planned notes as students ask questions, extend ideas, and even offer corrections—after all, no instructor is infallible! For the same reason, I engage students in worked examples, solving problems as a group in real time and highlighting potential mistakes as they arise. To accommodate different learning styles, I use props in the classroom. For example, I have shared a collection of different-sided dice prior to a lecture on sample spaces and outcomes—students especially enjoyed rolling the d120—and solicited volunteers to flip coins in order to build a skip list interactively as a class. The guided discovery process facilitated by students handling props establishes a foundation upon which I can build a memorable lecture. My approach incorporates a dimension of flexibility into lectures, allowing me to mould the content to the students rather than the other way around.

I believe my approaches have a positive effect on student learning, as evidenced by both the numeric ratings and the student comments I have received. Such feedback provides reassurance that I am achieving my goal of bringing meaning to theory. That being said, I still seek to improve various aspects of my teaching style. Currently, my priority is to identify areas where I can increase the use of technology in my lectures to facilitate active learning, including using software tools to illustrate concepts (e.g., constructing automata with JFLAP) or leading live-coding sessions during algorithm-focused lectures. In past courses, I have introduced students to the LaTeX document preparation system via an interactive workshop, and this was generally received well. I also plan to incorporate more small group activities and student-led lessons in my lectures, taking advantage of the often-smaller class sizes in theory courses to build a sense of community.

3 Teaching Responsibilities

The following lists outline all courses for which I had some kind of teaching responsibility, including as an instructor or guest lecturer, as an instructional apprentice (IA) or teaching assistant (TA), or as a curriculum developer. Appendix A contains course descriptions for all courses listed here. Appendices C–E contain examples of materials from a recent course I have taught.

All courses from Fall 2017 to present were held at Queen's University.
All courses from Fall 2015 to Spring 2017 were held at the University of Waterloo.

3.1 Course Instruction

Primary responsibilities include planning and delivering lectures, designing in-class activities, writing lecture notes, creating assignments and exams, coordinating teaching assistants, holding office hours, and handling administrative matters.

Winter 2019	CISC 203: Discrete Mathematics for Computing II 49 students, 1 instructor, 2 TAs, 3 lecture hours per week – Based on existing course design with development of new material.
Spring 2017	CS 240: Data Structures and Data Management 340 students, 3 instructors, 1 IA, 7 TAs, 3 lecture hours per week – Based on existing course design.

3.2 Guest Lecturing

Primary responsibilities include planning and delivering self-contained lectures on course material. While lecture topics were set by the instructor, I was free to determine the focus and format of individual lectures.

Fall 2019	CISC 203: Discrete Mathematics for Computing II 246 students, 3 lectures, 1 hour each – Delivered lectures on graph embeddings and planar graphs.

3.3 Instructional Apprenticeships

At the University of Waterloo, the role of instructional apprentice recognizes the advanced skills of some graduate students. IAs are selected by instructors on the basis of TA excellence. Primary responsibilities include leading tutorials, coordinating teaching assistants, and creating solution sets and marking schemes.

Fall 2016	CS 234: Data Types and Structures
Spring 2016	CS 240: Data Structures and Data Management 121 students, 2 sections, 1 tutorial hour per week per section – Presented and worked through example problems related to course material.

3.4 Teaching Assistantships

Primary responsibilities include holding office hours, marking assignments and exams, and proctoring assessments. In recent terms, responsibilities have also included coordinating junior teaching assistants, creating solution sets and marking schemes, and adapting content for an online environment.

Winter 2021	CISC/CMPE 223: Software Specifications (online)
Fall 2020	CISC 203: Discrete Mathematics for Computing II (online)
Winter 2020	CISC/CMPE 223: Software Specifications (partially online)
Fall 2019	CISC 203: Discrete Mathematics for Computing II

Fall 2018	CISC 462: Computability and Complexity
Winter 2018	CISC/CMPE 223: Software Specifications
Fall 2017	CISC 462: Computability and Complexity
Winter 2017	CS 462/662: Formal Languages and Parsing
Winter 2016	CS 240: Data Structures and Data Management
Fall 2015	CS 234: Data Types and Structures

3.5 Curriculum Development

Winter 2019 CISC 203: Discrete Mathematics for Computing II
– Wrote ∼110 pages of lecture notes with illustrations, proofs, and worked examples.

Spring 2017 CS 240: Data Structures and Data Management
– Wrote a ∼300-page teaching guide based on existing lecture slides.

3.6 Relevant Committee Service

2014 – 2015 Curriculum Committee, Department of Computer Science, University of Western Ontario
– Primarily tasked with undertaking a curriculum mapping process to align undergraduate course learning outcomes with CIPS CSAC graduate attributes.

4 Evidence of Teaching Effectiveness

4.1 Summary

The following table summarizes my "effectiveness rating" as an instructor for past courses. Department means are provided when available. Additional data are provided in Appendix B.

Term	**Course**	**Effectiveness Rating**	**Department Mean**
Winter 2019	CISC 203	4.7/5.0	3.7/5.0
Spring 2017	CS 240	4.3/5.0	—

4.2 Honours and Awards

2020 TA/TF Excellence Award, Queen's Society of Graduate & Professional Students (SGPS)
– University-level award given to recognize the outstanding contributions of a teaching assistant/teaching fellow to the SGPS and Queen's community. One recipient per year.
– First student from the School of Computing to receive this award.

2018 Excellence in Teaching Assistance Award, School of Computing, Queen's University
– Department-level award given to a teaching assistant in a computing course who went above and beyond in their duties. One recipient per year.

4.3 Representative Feedback from Students

> My name is [...]. I was a Discrete Mathematics II student of yours this past semester. Your enthusiasm for the subject matter was infectious, and inspired me to consider Post-Graduate research in Pure Mathematics. I am reaching out to thank you, and wish you the best of luck, whatever your future may hold.
> *(Email from former CISC 203 student, Winter 2019)*

> Prof. Smith is very diligent and clear when he teaches. He is organized, and his lectures have a flow/story to them that make the material easy to understand. The assignment problems stimulate my creative-thinking skills. He is also very helpful during office hours. This course could be very boring and difficult without him.
> *(CISC 203 feedback, Winter 2019)*

> I liked how genuinely interested Prof. Smith was in teaching the course. His enthusiasm makes coming to this class the highlight of my day. His lecture notes both in-person and online are in-depth and I know that if I have to miss a class, the online notes will prepare me for the next assignment.
> *(CISC 203 feedback, Winter 2019)*

> I was originally skeptical about having a grad student as an instructor, but Taylor ended up being one of the better instructors I've had. It may be a symptom of him sympathizing more with us undergrads than some tenured profs do, but he targeted his explanations at precisely the right level, and actually answered low-level questions when students were having a tough time understanding the content, never blowing off questions with the equivalent of "I shouldn't have to explain that, you should be smart enough to figure that out," which I've seen from several professors before.
> *(CS 240 feedback, Spring 2017)*

4.4 Representative Feedback from Colleagues

> Taylor was my choice for head TA for the CISC 223 course. I made this choice when I heard that he has had stellar feedback and standing ovations from students in CISC 203 in Winter 2019. I run my courses by democratic council to avoid power dynamics as well as a redundancy layer for important choices. Taylor has been vital in this process. He has provided great insight for some choices, including midterm and final exam design. He is available for discussion, and I have heard great feedback from students that go to office hours. He also volunteered to do a test run of the midterm; it was a welcome surprise.
> *(CISC/CMPE 223 instructor, Winter 2020)*

5 Professional Development

5.1 Professional Development in University Teaching and Learning Program, Queen's University

The Queen's University Professional Development in University Teaching and Learning program is intended for students and fellows who have an interest in developing themselves in areas of university teaching and learning. The program consists of five components that are specific to each of the areas of university teaching and learning.

As a participant in this program, I gained an understanding of foundational issues in university teaching and used my teaching experience to articulate my beliefs and attitudes toward these issues. I also became acquainted with the scholarship of teaching and learning within the context of my subject area. Finally, I applied Universal Design for Learning strategies to course material I created in the past to make it more accessible and inclusive.

Appendix F contains a research prospectus and annotated bibliography I developed as part of the "Scholarship in Teaching and Learning" component of this program.

Workshops completed	Foundations in Teaching and Learning Practical Experience Educational Leadership Scholarship in Teaching and Learning Accessible Teaching and Learning

5.2 Certificate in Professional Development, Queen's University

The Queen's University Certificate in Professional Development consists of a series of workshops and seminars to support the academic, personal, and professional success of graduate students and postdoctoral fellows. Participants complete workshops in the areas of Health, Wellness, and Community; Research Skills; Communication; Management and Leadership Development; Career Building; and Setting Ideas in Motion.

While the components of this certificate were not solely focused on teaching, my experience in the program allowed me to develop skills for providing effective leadership, enhanced my awareness of the principles of teaching in a diverse environment, and improved my ability to communicate and contextualize advanced topics in my subject area.

5.3 Fundamentals of University Teaching Program, University of Waterloo

The University of Waterloo Fundamentals of University Teaching program supports graduate students in developing their knowledge and skills as university TAs and instructors. Participants attend a minimum of six teaching workshops and lead three small-group microteaching sessions.

As a participant in this program, I gained experience in teaching topics from my subject area to a small non-expert audience, developed techniques for organizing and structuring lectures according to pre-established learning objectives, and experimented with a variety of active learning and student-centred teaching methods. I also completed two self-directed modules on the theory of online learning, where I developed strategies for facilitating online learning effectively in both fully-online and hybrid models.

Workshops completed	Effective Lesson Plans Giving and Receiving Feedback Teaching Online – Basic Skills Teaching Online – Advanced Skills Classroom Delivery Skills Teaching Methods Teaching STEM Tutorials

A Course Descriptions

Queen's University

CISC 203: Discrete Mathematics for Computing II

Proof methods. Combinatorics: permutations and combinations, discrete probability, recurrence relations. Graphs and trees. Boolean and abstract algebra.

CISC/CMPE 223: Software Specifications

Introduction to techniques for specifying the behaviour of software, with applications of these techniques to design, verification and construction of software. Logic-based techniques such as loop invariants and class invariants. Automata and grammar-based techniques, with applications to scanners, parsers, user-interface dialogs and embedded systems. Computability issues in software specifications.

CISC 462: Computability and Complexity

Turing machines and other models of computability such as λ-recursive functions and random-access machines. Undecidability. Recursive and recursively enumerable sets. Church-Turing thesis. Resource-bounded complexity. Complexity comparisons among computational models. Reductions. Complete problems for complexity classes.

University of Waterloo

CS 234: Data Types and Structures

Top-down design of data structures. Using representation-independent data types. Introduction to commonly used data types, including lists, sets, mappings, and trees. Selection of data representation.

CS 240: Data Structures and Data Management

Introduction to widely used and effective methods of data organization, focusing on data structures, their algorithms, and the performance of these algorithms. Specific topics include priority queues, sorting, dictionaries, data structures for text processing.

CS 462/662: Formal Languages and Parsing

Languages and their representations. Grammars –Chomsky hierarchy. Regular sets and sequential machines. Context-free grammars –normal forms, basic properties. Pushdown automata and transducers. Operations on languages. Undecidable problems in language theory. Applications to the design of programming languages and compiler construction.

B Course Evaluations

Winter 2019 – CISC 203: Discrete Mathematics for Computing II

Response rate: 35 / 49 (71.4%)

Instructor is an effective teacher:

Strong agree	**18**
Agree	**14**
Neutral	**3**
Disagree	**0**
Strong disagree	**0**
Not applicable	**0**

This is an excellent course:

Strong agree	25
Agree	9
Neutral	0
Disagree	0
Strong disagree	0
Not applicable	0

Instructor was sensitive to student needs/interests:

Strong agree	22
Agree	7
Neutral	2
Disagree	0
Strong disagree	0
Not applicable	3

Student interest in subject was stimulated by course:

Strong agree	17
Agree	11
Neutral	6
Disagree	1
Strong disagree	0
Not applicable	0

Instructor presented material clearly:

Strong agree	22
Agree	12
Neutral	0
Disagree	0
Strong disagree	0
Not applicable	0

Instructor was available outside of class:

Strong agree	27
Agree	4
Neutral	1
Disagree	0
Strong disagree	0
Not applicable	1

Instructor showed genuine concern for students:

Strong agree	24
Agree	9
Neutral	1
Disagree	0
Strong disagree	0
Not applicable	0

Course was well-organized:

Strong agree	22
Agree	12
Neutral	0
Disagree	0
Strong disagree	0
Not applicable	0

Spring 2017 – CS 240: Data Structures and Data Management

Response rate: 64 / 140 (45.7%)

Overall effectiveness of instructor:

Excellent	**30**
Good	**26**
Satisfactory	**3**
Unsatisfactory	**1**
Very poor	**1**
No opinion	**0**

Organization and coherence of lectures:

Excellent	27
Good	28
Satisfactory	6
Unsatisfactory	0
Very poor	1
No opinion	1

Level of instructor's explanations:

Too high	2
Somewhat high	8
Just right	44
Somewhat low	5
Too low	2
No opinion	0

Instructor's treatment of student questions:

Excellent	29
Good	25
Satisfactory	5
Unsatisfactory	0
Very poor	1
No opinion	1

Effectiveness of instructor's visual presentation:

Excellent	30
Good	23
Satisfactory	5
Unsatisfactory	1
Very poor	1
No opinion	1

Effectiveness of instructor's oral presentation:

Excellent	34
Good	23
Satisfactory	1
Unsatisfactory	1
Very poor	1
No opinion	1

Instructor availability outside of class:

Available	17
Usually	13
Sometimes	2
Rarely	0
Unavailable	0
Didn't seek help	28

Interest in the course:

Very interesting	22
Interesting	35
Not interesting	3
No opinion	2

C Recent Course Website

Taylor J. Smith | Courses

CISC 203: Discrete Mathematics for Computing II (Winter 2019)

Course Description

Proof methods. Combinatorics: permutations and combinations, discrete probability, recurrence relations. Graphs and trees. Boolean and abstract algebra.

See the course outline for more information.

Course Details

Lecture Time/Place

Tuesday, 12:30pm–1:30pm; Thursday, 11:30am–12:30pm; Friday, 1:30pm–2:30pm.

All lectures are held in Goodwin Hall, room 254.

Textbook

E. Scheinerman, *Mathematics: A Discrete Introduction*. Cengage Learning, 3rd edition, 2012.

The textbook is available for sale at the campus bookstore. Course notes will also be provided for each lecture.

Marking Scheme

- 50% assignments (12.5% per assignment)
- 25% midterm
- 25% final exam

You must pass the final exam to pass the course.

News

- Apr. 12: The final exam will be written on Apr. 26. See the Exams section below for details. A review session will be held on Apr. 25 at 1pm in Goodwin Hall, room 230. Office hours for the week of the final exam are as follows:
  - Taylor: Apr. 23, 1pm–3pm and Apr. 25, 3pm–4pm
- Apr. 5: Notes on Boolean and abstract algebra have been posted. The seventh problem set has also been posted.
- Mar. 31: The second half of the notes on trees has been posted.
- Mar. 25: The first half of the notes on trees has been posted. The sixth problem set has also been posted.
- Mar. 19: Assignment 4 has been posted. It is due by the beginning of class on Apr. 5.
- Mar. 18: The third part of the notes on graph theory has been posted.
- Mar. 12: The first two parts of the notes on graph theory have been posted. The fifth problem set has also been posted.
- Mar. 4: Assignment 3 has been posted. It is due by the beginning of class on Mar. 21.
- Feb. 25: The midterm will be written this week during the Feb. 28 lecture. See the Exams section below for details. The Feb. 26 lecture will serve as a review session. Office hours this week are as follows:

- Taylor: Feb. 26, 1:30pm–3:30pm (note the extended time)
- Xavier: Feb. 26, 8:30am–9:30am (note the changed date/time)
- Leonard: Feb. 27, 10:30am–11:30am

- Feb. 14: Notes on recurrence relations have been posted. The fourth problem set has also been posted.
- Feb. 12: Due to inclement weather, both the lecture and the office hour are cancelled today. Weather permitting, a makeup office hour will be held Feb. 13 at 1:30pm. The due date for Assignment 2 is not affected.
- Feb. 6: The second half of the notes on discrete probability has been posted. These notes contain material that was not covered in lecture (variance and probability distributions). This material is included for enrichment and will not be tested.
- Feb. 1: The first half of the notes on discrete probability has been posted. The third problem set has also been posted.
- Jan. 28: Assignment 2 has been posted. It is due by the beginning of class on Feb. 14.
- Jan. 25: The second half of the notes on combinatorics has been posted.
- Jan. 22: The first half of the notes on combinatorics has been posted. The second problem set has also been posted.
- Jan. 14: Assignment 1 has been posted. It is due by the beginning of class on Jan. 31.
- Jan. 11: Notes on proof techniques have been posted. The first problem set has also been posted. Problem sets are optional and not for marks, but they are great for some additional practice.
- Jan. 7: Welcome to the course! The first lecture is on Tuesday, Jan. 8.

Lectures

Week	Notes	Readings
1	Introduction, Review, Proof Techniques	Scheinerman, 1.2–1.6, 4.20–4.22 (Problem set)
2	Combinatorics: basic techniques, permutations	Scheinerman, 2.8, 3.19, 5.25, 5.27 (Problem set)
3	Combinatorics (cont'd): combinations, binomial theorem	Scheinerman, 3.17–3.18
4	Discrete Probability: definitions, properties, conditionals, RVs	Scheinerman, 6.30–6.33 (Problem set)
5	Discrete Probability (cont'd): RVs, expectation; Recurrences	Scheinerman, 6.33–6.34, 4.23 (Problem set)
6	Recurrences (cont'd), Algorithm Analysis (bonus topic)	Scheinerman, 4.23 (optional: 5.29)
—	Reading week	—
7	Review session and midterm; Graphs: definitions	Scheinerman, 9.47–9.48 (Problem set)
8	Graphs (cont'd): representations, paths/circuits, connectivity	Scheinerman, 9.49, 9.51
9	Graphs (cont'd): isomorphisms, planarity, colouring	Scheinerman, 9.52–9.53
10	Trees: definitions, properties, representations	Scheinerman, 9.50 (Problem set)
11	Trees (cont'd): traversals, spanning trees	Scheinerman, 9.50
12	Boolean and Abstract Algebra	Scheinerman, 1.7, 8.40 (Problem set)

Assignments

- Assignment 1, due January 31, 2019
- Assignment 2, due February 14, 2019
- Assignment 3, due March 21, 2019
- Assignment 4, due April 5, 2019

Assignments are due at the beginning of class on the due date. Late assignments will be accepted up to the beginning of the first class following the due date. Late assignments are subject to a penalty of 10% deducted from the earned mark.

Exams

Midterm

Date: February 28, 2019
Time: 11:30am–12:20pm
Place: Goodwin Hall, room 247 (last names A–M) and Goodwin Hall, room 254 (last names N–Z)
Content: All material from lectures up to and including Week 6

Final Exam

Date: April 26, 2019

Time: 7:00pm–9:00pm
Place: Mitchell Hall, gym 5
Content: All material from all lectures

Personnel

Instructor

Taylor J. Smith
Email: tsmith [at] cs [dot] queensu [dot] ca
Office: Goodwin Hall, room 235
Office hours: Tuesday, 1:30pm–2:30pm; Thursday, 12:30pm–1:30pm

Teaching Assistants

Xavier McMaster-Hubner
Email: 17xjem [at] queensu [dot] ca
Office: Goodwin Hall, room 241
Office hours: Monday, 10:30am–11:30am

Leonard Zhao
Email: 16lz1 [at] queensu [dot] ca
Office: Goodwin Hall, room 241
Office hours: Wednesday, 10:30am–11:30am

D Recent Course Outline

Queen's University
School of Computing

CISC 203: Discrete Mathematics for Computing II
Course Outline
Winter 2019

1 Course Description

Proof methods. Combinatorics: permutations and combinations, discrete probability, recurrence relations. Graphs and trees. Boolean and abstract algebra.

2 Learning Outcomes

A course learning outcome is a brief statement of a skill, competency, or attitude a successful student will achieve by the end of a course. The following list of learning outcomes for CISC 203 is provided by the School of Computing. (`http://www.cs.queensu.ca/students/undergraduate/outcomes/CLO.php`)

- Critique and construct moderately sophisticated mathematical arguments such as proof by contradiction, proof by induction, proof by minimal counterexample, counting arguments, and recognition of orderings.
- Apply discrete mathematical tools and models such as graph theory, probability, group theory and modular arithmetic to problems such as modelling relational data and networks, scheduling and resource allocation, network design, predicting expected performance, motion planning, cryptography.
- Apply basic discrete probability techniques to computational tasks.
- Build a foundation for further learning by exposure to multiple computer languages, development tools, and methodologies.

3 Personnel

3.1 Instructor

Taylor J. Smith

- Email: `tsmith@cs.queensu.ca`
- Office: Goodwin Hall, room 235
- Office hours: Tuesday, 1:30pm–2:30pm; Thursday, 12:30pm–1:30pm

3.2 Teaching Assistants

Xavier McMaster-Hubner
- Email: `17xjem@queensu.ca`
- Office: Goodwin Hall, room 241
- Office hours: Monday, 10:30am–11:30am

Leonard Zhao
- Email: `16lz1@queensu.ca`
- Office: Goodwin Hall, room 241
- Office hours: Wednesday, 10:30am–11:30am

4 Course Details

4.1 Lecture Time/Place

- Tuesday, 12:30pm–1:30pm
- Thursday, 11:30am–12:30pm
- Friday, 1:30pm–2:30pm

All lectures are held in Goodwin Hall, room 254.

4.2 Textbook

E. Scheinerman, *Mathematics: A Discrete Introduction*. Cengage Learning, 3rd edition, 2012.

4.3 Prerequisites

This course requires [CISC 102 (Discrete Mathematics for Computing I) or MATH 110 (Linear Algebra)] and CISC 121 (Introduction to Computing Science I) as prerequisites, each with a minimum mark of C–.

This course is a prerequisite for CISC 235 (Data Structures), CISC 322 (Software Architecture), CISC 333 (Introduction to Data Mining), and CISC 365 (Algorithms I).

5 Evaluations

5.1 Marking Scheme

- 50% assignments (12.5% per assignment)
- 25% midterm
- 25% final exam

You must pass the final exam to pass the course.

The marking scheme will not be adjusted for individual students, with the exception of students who miss an evaluation due to illness or other extenuating circumstances.

5.2 Assignments

There will be four written assignments in this course. Assignments will be due at the beginning of class on the due date. Only paper assignments will be accepted; email copies will not be accepted.

Assignment solutions must be based on individual work. See Section 6 for more information about academic integrity.

Late assignments will be accepted up to the beginning of the first class following the due date. Assignments submitted later than this time will not be accepted. Late assignments are subject to a penalty of 10% deducted from the earned mark.

If you must miss an assignment due to illness or other extenuating circumstances, please contact the instructor prior to the assignment due date/time. See Section 5.5 for more information about academic considerations and accommodations.

If you have questions about assignment marking, please contact the teaching assistant within one week from the time you received the marked assignment. All assignment marks are considered final after one week has passed from the time you received the marked assignment.

5.3 Midterm and Final Exams

The midterm exam will be held in class during the usual lecture time.

If you have questions about midterm exam marking, please contact the instructor within one week from the time you received the marked midterm exam. All midterm exam marks are considered final after one week has passed from the time you received the marked midterm exam.

The final exam will be held during the examination period in April. It will be scheduled by the Examinations Office. More details are listed in Section 5.4.

If you require accommodations for the midterm or final exam, please follow the procedures listed in Section 5.5.

5.4 Location and Timing of Final Exams

Arts and Science Regulation 8.2.1 states

> The final examination in any class offered in a term or session (including Summer Term) must be written on the campus on which it was taken, at the end of the appropriate term or session at the time scheduled by the Examinations Office.

The exam period is listed in the key dates prior to the start of the academic year in the Faculty of Arts and Science Academic Calendar and on the Office of the University Registrar's webpage. A detailed exam schedule for the Fall Term is posted before the Thanksgiving holiday; for the Winter Term it is posted the Friday before Reading Week, and for the Summer Term the window of dates is noted on the Arts and Science Online syllabus prior to the start of the course. Students should delay finalizing any travel plans until after the examination schedule has been posted. Exams will not be moved or deferred to accommodate employment, travel/holiday plans or flight reservations.

5.5 Academic Considerations and Accommodations

Academic considerations and accommodations are two different mechanisms for helping students in extenuating circumstances. If you have extenuating circumstances for missing a midterm or assignment deadline, or for long-term issues, see the Student Wellness website.
(`http://www.queensu.ca/studentwellness/resources/students-extenuating-circumstances`)

Queen's University is committed to achieving full accessibility for persons with disabilities. Part of this commitment includes arranging academic accommodations for students with disabilities to ensure they have an equitable opportunity to participate in all of their academic activities. If you are a student with a disability and think you may need accommodations, you are strongly encouraged to contact Student Wellness Services (SWS) and register as early as possible. For more information, including important deadlines, please visit the Student Wellness website.
(`http://www.queensu.ca/studentwellness/resources/students-extenuating-circumstances`)

The Senate Policy on Academic Consideration for Students in Extenuating Circumstances was approved in April 2017. Queen's University is committed to providing academic consideration to students experiencing extenuating circumstances that are beyond their control and which have a direct and substantial impact on their ability to meet essential academic requirements. Each Faculty has developed a protocol to provide a consistent and equitable approach in dealing with requests for academic consideration for students facing extenuating circumstances. Arts and Science undergraduate students please consult the Faculty of Arts and Science protocol and the portal where they submit a request. Students in other Faculties and Schools should refer to the protocol for their home Faculty.

5.5.1 Academic Considerations and Accommodations for Tests

Some of your instructors may participate in central administration of special tests (including midterms and quizzes) for students with accommodations (other than "use of computer", which is handled by the exams office); the same process will apply for make-up tests requested via the Arts and Science academic considerations portal. Although we will strive to ensure you are accommodated to the standards in your form, if we do not have, at minimum, 10 working days' notice, we can't guarantee that your accommodation needs will be fully met. Shorter notice may be possible with academic considerations for students who don't have accommodations that complicate scheduling, but we can't guarantee it.

Students' accommodation tests will be booked either at the same time as the rest of the class, or as soon after as possible, according to their accommodation requirements. Make-up tests for academic considerations will similarly be booked as soon as possible. Your class schedule will be taken into account when making the booking; due to the expected volume of requests and bookings, we will not be able to book according to the student's preference. Once we have finalized the booking and arranged for a proctor, you will be contacted by email.

IMPORTANT: Should you elect to write your test with the rest of the class, instead of writing the accommodated test that was booked for you, we require two working days' notice so that we can cancel your room booking, and notify the proctor that was hired to invigilate your test.

All computer accommodations are handled by the Exams Office. They require 10 working days' notice to make these arrangements.

If you have questions or concerns, or would like to discuss your case, please feel free to contact us at `accommodation@cs.queensu.ca`, 613-533-6050, or drop by the School of Computing main office in Goodwin Hall, room 557 (8am–12pm and 1pm–4pm).

6 Academic Integrity

Queen's students, faculty, administrators and staff all have responsibilities for supporting and upholding the fundamental values of academic integrity. Academic integrity is constituted by the five core fundamental values of honesty, trust, fairness, respect and responsibility (see `www.academicintegrity.org`) and by the quality of courage. These values and qualities are central to the building, nurturing and sustaining of an academic community in which all members of the community will thrive. Adherence to the values expressed through academic integrity forms a foundation for the "freedom of inquiry and exchange of ideas" essential to the intellectual life of the University.

Students are responsible for familiarizing themselves with and adhering to the regulations concerning academic integrity. General information on academic integrity is available at Integrity@Queen's University, along with Faculty or School specific information. Departures from academic integrity include, but are not limited to, plagiarism, use of unauthorized materials, facilitation, forgery and falsification. Actions which contravene the regulation on academic integrity carry sanctions that can range from a warning, to loss of grades on an assignment, to failure of a course, to requirement to withdraw from the university.

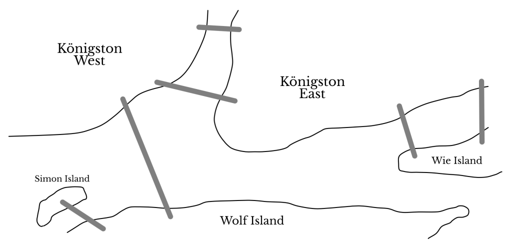
E Sample Course Materials

Queen's University
School of Computing

CISC 203: Discrete Mathematics for Computing II
Lecture 5: Graph Theory
Winter 2019

1 Traveling Around Königston

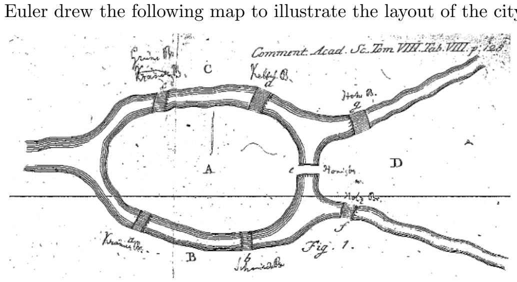
Imagine that, one day, you take a trip to the City of Königston. (This city looks a lot like Kingston, but it has a large German population, hence the name.) Königston consists of five regions: Königston West, Königston East, Wolf Island, Wie Island, and Simon Island. They are arranged as follows:



To move between these five regions, you can take either a bridge or a ferry. Since you want to get the full Königston experience, but you don't want to stay out too late, you decide to cross each bridge and ride each ferry exactly once.

During your trip, the local government announces a new ferry service between Wolf Island and Wie Island. Naturally, you feel the need to redo your tour. However, with this new ferry, it is no longer possible for you to cross each bridge and ride each ferry exactly once! How could such a small change have such a big effect?

In 1736, the Swiss mathematician Leonhard Euler thought about a remarkably similar problem called the "Bridges of Königsberg". The downtown area of the City of Königsberg (now known as Kaliningrad, Russia) is divided into four parts by a river, and each of these parts are connected to one another by a total of seven bridges. In his paper, Euler drew the following map to illustrate the layout of the city:



Interestingly, Euler found that, no matter what path you take across the bridges, you cannot cross each of the bridges exactly once. With this simple puzzle, Euler inadvertently kicked off the entire study of **graph theory**—something that affects both mathematicians and tourists to this day.

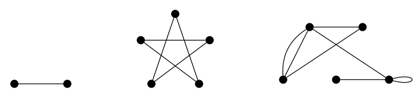
2 Graphs

At its core, a **graph** is not a complicated concept. In fact, graphs are only made up of two sets.

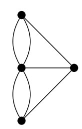
Definition 1 (Graph)**.** A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E, where each element of E is a pair $\{u, v\}$ of vertices $u, v \in V$.

In other words, a graph is just a set of elements (which we call **vertices**) and a set of relations that join those elements (which we call **edges**).

Example 2. Each of the following are graphs:



Example 3. We can model the Bridges of Königsberg problem with the following graph:



2.1 Properties of Graphs

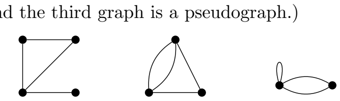
You may have noticed that Definition 1 is worded in a very general way, and it allows for strange things that might make our graphs complicated, like multiple edges between the same vertices or edges joining a vertex to itself (called **loops**). If we like, we can refine our definition to consider restricted graphs that don't allow for such strange things. We call such graphs **simple**; these are the kinds of graphs we will be dealing with most of the time.

Definition 4 (Simple graph)**.** A graph $G = (V, E)$ is simple if,

1. given two elements $\{u_1, v_1\}, \{u_2, v_2\} \in E$, both $u_1 \neq u_2$ and $v_1 \neq v_2$ (no multiple edges); and
2. if $\{u, v\} \in E$, then $u \neq v$ (no loops).

In the literature, graphs with multiple edges are called **multigraphs** and graphs with both multiple edges and loops are called **pseudographs**, but we will not use such terminology here.

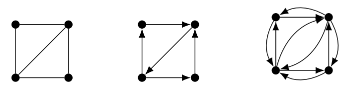
Example 5. The first graph is a simple graph. Neither the second nor third graphs are simple graphs. (The second graph is a multigraph and the third graph is a pseudograph.)



Both Definitions 1 and 4 assume that the set of edges E consists of unordered pairs, so if an edge exists between vertices u and v, then another edge implicitly exists between v and u. If we instead require that E consists of ordered pairs, then the existence of an edge between vertices u and v does not necessarily imply the existence of an edge between v and u. We say that such graphs are **directed**, because the direction of each edge in the set E matters when we move between vertices. Directed graphs are occasionally referred to by the shorthand name of "digraphs".

As you would expect, we say that graphs where the direction of edges does not matter are **undirected**.

Example 6. Out of the following graphs, the first graph is undirected and both the second and third graphs are directed. The third graph is the directed equivalent of the first graph.

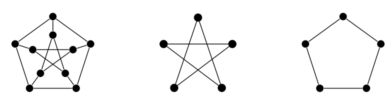


Finally, if we have two graphs $G = (V, E)$ and $H = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$, then we say that H is a **subgraph** of G. In other words, a subgraph is a copy of G with vertices or edges (or both) removed. Note that if we remove a vertex, then we cannot leave behind any edges that touched the removed vertex.

If H is obtained from G by deleting only edges, then H is sometimes called a **spanning subgraph** of G. Likewise, if H is obtained from G by deleting only vertices, then H is an **induced subgraph** of G. For simplicity's sake, we will not use either of these terms; we will only refer to subgraphs in general.

Subgraphs are particularly useful when we discuss certain problems in graph theory, since we may be able to reduce a given graph to a subgraph that is easier to use or understand.

Example 7. The leftmost graph, known as the **Petersen graph**, is used frequently as an example or counterexample when proving results in graph theory. We will see the Petersen graph again later in these notes, but for now, observe that the Petersen graph contains the two rightmost graphs as subgraphs.



2.2 Properties of Vertices and Edges

Now that we have established terminology for talking about graphs in general, let's look at some terminology for talking about particular properties of a graph. In this section, assume that we are given an arbitrary graph $G = (V, E)$.

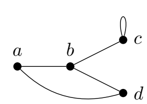
Given vertices $u, v \in V$, if $\{u, v\} \in E$ (that is, if an edge exists between u and v), then we say that u and v are **adjacent**, and we also say that the edge $\{u, v\}$ is **incident** to the vertices u and v. The set of all vertices adjacent to some vertex u is called the **neighbourhood** of u.

We say that a vertex u is of **degree** k if it is adjacent to k other vertices in the graph. We denote the degree of a vertex u by $\deg(u)$. We may formally define the degree of a vertex u as

$$\deg(u) = |\{v \in V \mid \{u, v\} \in E\}| .$$

Remark. In the case where a vertex u has an undirected loop, we must add two to the degree of u. The reason we do this is because if we rewrite the undirected loop as two directed loops—one for each direction—then u ends up being adjacent to itself twice.

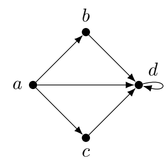
Example 8. In the following graph, we see that vertex a is adjacent to vertices b and d; vertex b is adjacent to vertices a, c, and d; vertex c is adjacent to vertices b and c; and vertex d is adjacent to vertices a and b. From this, we see that $\deg(a) = 2$, $\deg(b) = 3$, $\deg(c) = 3$ (since the loop adds two to the degree), and $\deg(d) = 2$.



For directed graphs in particular, we can define specific kinds of degrees. Namely, the **indegree** of a vertex u (denoted $\deg^{-}(u)$) is the number of vertices with edges that lead to u, and the **outdegree** of u (denoted

$\deg^+(u)$) is the number of vertices with edges that lead away from u. Intuitively, we can think of the indegree of u as the number of arrow heads touching u and, similarly, we can think of the outdegree of u as the number of arrow tails touching u.

Example 9. In the following graph, we see that $\deg^-(a) = 0$, $\deg^+(a) = 3$, $\deg^-(b) = \deg^+(b) = 1$, $\deg^-(c) = \deg^+(c) = 1$, $\deg^-(d) = 4$, and $\deg^+(d) = 1$.



If each vertex of an undirected graph G has the same degree, then G is a **regular** graph. A regular graph G is sometimes called ***k*-regular** if the degree of each vertex is k. A directed graph is regular only if the indegree and outdegree of each vertex are equal to one another.

Using what we know about vertices, edges, and degrees, we can now prove our first small result in graph theory: if we sum the degrees of every vertex in an undirected graph G, our total will be exactly twice the number of edges of G.

Lemma 10 (Handshake lemma)**.** *Given an undirected graph $G = (V, E)$,*

$$\sum_{u \in V} \deg(u) = 2 \cdot |E|.$$

Proof. Each edge $\{u, v\} \in E$ contributes one to the degrees of the vertices u and v. In the case where the edge is not a loop, adding one to both $\deg(u)$ and $\deg(v)$ double-counts that edge. In the case where the edge is a loop, adding two to $\deg(u)$ also double-counts that edge. Therefore, the summation of the degrees of all vertices in G will be exactly twice the number of edges in G. □

We call the previous result the "handshake lemma" because of the fun observation that, if n people shake hands with one another at a party, then exactly $2n$ hands will be shaken. (As an exercise, verify this at the next party you attend.)

There is an analogue to the handshake lemma for directed graphs, which relates the number of edges to both the indegree and the outdegree of each vertex. We will not prove it here, but we will state it out of interest.

Lemma 11 (Handshake lemma—directed graphs)**.** *Given a directed graph $G = (V, E)$,*

$$\sum_{u \in V} \deg^-(u) = \sum_{u \in V} \deg^+(u) = |E|.$$

Proof. Omitted. □

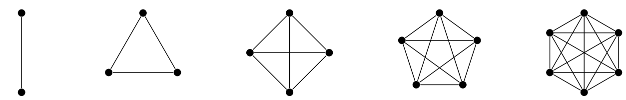
2.3 Special Graphs

Throughout our study of graph theory, we will see certain special graphs come up often in definitions, theorems, and examples. Let us look at a few of these special graphs now.

The **null graph** is the graph containing zero vertices and zero edges. The **singleton graph** is the graph containing one vertex and zero edges.

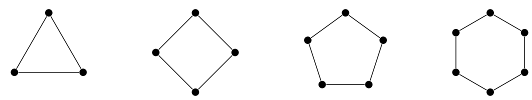
null graph singleton graph

The **complete graph** on n vertices, denoted K_n, consists of n vertices and $\frac{n(n-1)}{2}$ edges joining each vertex to every other vertex exactly once. (Note that the null and singleton graphs are equivalent to the complete graphs K_0 and K_1, respectively.)



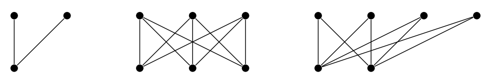
K_2 K_3 K_4 K_5 K_6

The **cycle graph** on $n \geq 3$ vertices, denoted C_n, consists of n vertices and n edges that join each vertex u_i to the vertex u_{i+1} for all $1 \leq i \leq n$. The vertex u_n is joined to the vertex u_1 to complete the cycle.



C_3 C_4 C_5 C_6

A **bipartite graph** is a graph whose vertex set V can be partitioned into two subsets V_1 and V_2 such that each edge joins a vertex in V_1 to a vertex in V_2. If V_1 contains m vertices and V_2 contains n vertices, and if each vertex in V_1 is joined to every vertex in V_2, then we say that the graph is a **complete bipartite graph**, denoted $K_{m,n}$.



$K_{2,1}$ $K_{3,3}$ $K_{4,2}$

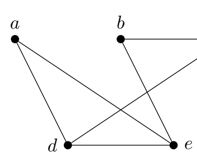
2.4 Representing Graphs

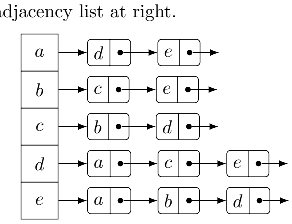
There are a number of methods of representing graphs apart from listing vertices and edges as explicit sets. Depending on how we are using graphs in an application, representing graphs using such methods could lead to ease-of-use or efficiency gains; for instance, storing vertices and edges in a data structure will result in faster access times compared to storing such data in separate locations on a disk.

The first method of representing graphs is known as an **adjacency list**. This method uses both an array data structure and a linked list data structure: vertices are represented using indices of the array, and for each array entry, the linked list stored at that index records all vertices adjacent to the vertex at that index.

Using adjacency lists, we can access vertices in constant time by indexing into the array, and we may determine all of the vertices adjacent to a given vertex by reading the elements of the linked list, which can be done in time proportional to the degree of the vertex.

Example 12. The graph at left can be represented by the adjacency list at right.





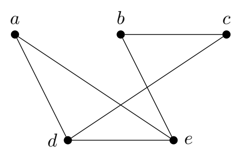
The second method of representing graphs is known as an **adjacency matrix**. As the name suggests, this method uses a matrix, or a two-dimensional array, to store vertex and edge data. Given a graph G with n vertices, the adjacency matrix of G (denoted $\mathbf{A}_G$) is an $n \times n$ matrix where the entry at row i and column j is equal to 1 when vertices u_i and u_j are adjacent and equal to 0 otherwise.

Remark. Adjacency matrices can represent loops on a vertex u_i by storing a 1 entry at position (i, i) of the matrix. We can modify our implementation of an adjacency matrix to handle graphs with multiple edges by storing the number of edges between vertices u_i and u_j at position (i, j) of the matrix.

If the graph G is undirected, then the associated adjacency matrix $\mathbf{A}_G$ will be symmetric; that is, the entries at positions (i, j) and (j, i) will be equal for all i and j. This is not always the case if G is a directed graph.

An adjacency matrix allows us to determine adjacent vertices by scanning a given row or column, which can be done in time proportional to the number of vertices, $|V|$. For graphs with many vertices and few edges, adjacency matrices perform worse than adjacency lists. However, adjacency matrices are easier to implement than adjacency lists.

Example 13. The graph at left (identical to the graph from Example 12) can be represented by the adjacency matrix at right.



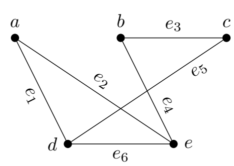
	a	b	c	d	e
a	0	0	0	1	1
b	0	0	1	0	1
c	0	1	0	1	0
d	1	0	1	0	1
e	1	1	0	1	0

The third method that we will see in this section also uses a matrix, but in a slightly different way. Instead of using rows and columns to represent vertices in a graph as we did with the adjacency matrix, we represent all vertices as rows and all edges as columns. The resulting structure is known as an **incidence matrix**. Given a graph G with n vertices and m edges, the incidence matrix of G (denoted $\mathbf{E}_G$) is an $n \times m$ matrix where the entry at row i and column j is equal to 1 if edge e_j is incident to vertex u_i and equal to 0 otherwise.

Remark. Just like with adjacency matrices, we can represent loops and multiple edges in an incidence matrix by making the appropriate changes to entries of the matrix. If we want to represent directed edges, we may use positive or negative entries depending on whether an edge is pointing toward or away from a given vertex.

In order to determine adjacent vertices in an incidence matrix, we must scan both the edge columns (to determine edges incident to our chosen vertex) and the vertex rows (to determine other vertices to which our chosen vertex is adjacent). This can be done in time proportional to the size of the matrix, $|V| \cdot |E|$, making it extremely inefficient compared to our previous methods. However, incidence matrices have their advantages; as an example, incidence matrices can distinguish multiple edges, while adjacency matrices cannot.

Example 14. The graph at left (identical to the graph from Example 12 with added edge labels) can be represented by the incidence matrix at right.



	e_1	e_2	e_3	e_4	e_5	e_6
a	1	1	0	0	0	0
b	0	0	1	1	0	0
c	0	0	1	0	1	0
d	1	0	0	0	1	1
e	0	1	0	1	0	1

Finally, keep in mind that this section is not exhaustive: there are many other methods to represent graphs, and certain methods lend themselves better to certain graphs. However, the three methods presented here are among the most commonly used methods of graph representation.

Queen's University
School of Computing

CISC 203: Discrete Mathematics for Computing II
Assignment 2
Due February 14, 2019 at 11:30am

[5 marks] 1. The Online Encyclopedia of Integer Sequences (`https://oeis.org`) is a valuable computer resource for discovering and exploring patterns or hidden connections in integer sequences. For instance, sequence A000040 corresponds to the sequence of prime numbers, and sequence A000045 corresponds to terms of the Fibonacci sequence. In this problem, we will investigate a sequence that arises from partitioning the elements of a particular set.

Let $p(n)$ denote the number of ways to partition the set $S = \{1, 2, \ldots, 2n\}$ into n subsets each of size 2.

(a) Determine $p(1)$ and $p(2)$ directly.

(b) Develop a formula to calculate $p(n)$ for all $n \geq 2$.

(c) Visit `https://oeis.org` and enter a few terms of the sequence $p(n)$ as given by your formula. Which sequence is generated by your formula? Give the name and number of the sequence. (You will need to enter four or five terms to narrow your search sufficiently.)

[5 marks] 2. Analyze the following probabilistic argument. Is it correct? Explain why or why not.

> A member of the tech support staff is updating equipment in the server room. The probability of successfully updating the equipment is $1/3$ and each update attempt is independent. The staff member will make three attempts to update the equipment before quitting. Since the probability of a successful update is $1/3 = 0.33\ldots$, the probability of a successful update after three attempts is $3(1/3) = 0.99\ldots = 1$, so the staff member is certain to succeed in updating the equipment.

[5 marks] 3. In this question, we will consider the following problem in computational complexity theory:

<u>MAX-SAT</u> Given: a Boolean formula in conjunctive normal form Determine: an assignment of truth values that maximizes the number of true clauses in the formula

The MAX-SAT (maximum satisfiability) problem is similar to the Boolean satisfiability problem SAT, except instead of finding an assignment that makes every clause true, we want to find an assignment that maximizes the number of true clauses. Thus, MAX-SAT is a generalization of SAT.

As an example, consider the formula $(x_0 \vee x_1) \wedge (x_0 \vee \neg x_1) \wedge (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$. This formula is in conjunctive normal form; each clause contains only variables and $\vee$s, and all clauses are joined only by $\wedge$s. There is no way to satisfy all four clauses of this formula at once, but it is possible to satisfy three out of the four clauses.

(a) Suppose a formula in conjunctive normal form consists of n variables, and we assign truth values to each variable randomly by flipping a fair coin. If we assign T after flipping tails and F after flipping heads, what is the probability of each possible assignment of truth values to the n variables?

(b) Suppose a formula in conjunctive normal form is such that each clause is a disjunction of exactly two distinct variables or their negations; that is, each clause is of the form $(\_ \vee \_)$, where the blank spaces contain variables. What is the probability that a given clause is true, if we assign truth values randomly by flipping a fair coin as in part (a)?

(c) Suppose a formula in conjunctive normal form consists of c clauses as defined in part (b). What is the expected number of true clauses in the formula, if we assign truth values randomly by flipping a fair coin as in part (a)?

F Sample Scholarship Materials

Scholarship of Teaching and Learning
Research Prospectus
Taylor J. Smith

Courses in theoretical computer science are unlike most traditional computer science courses at the university level; they are, in fact, more akin to a course in pure or applied mathematics. In a theoretical computer science lecture, both students and instructors are unlikely to do any coding, and neither party may even be required to use a computer at all! Unfortunately, the inclination of the typical computer science student is to avoid courses that are math-heavy, and as a result, theory courses are often considered by students to be the least popular courses offered by a department.

As an undergraduate student, my first course in theoretical computer science was led by an instructor who relied on PowerPoint slides made by the author of the course textbook. As a result, I felt that neither I nor the instructor had any "real" connection to the material being taught in the course, and my attention waned during each lecture. Prior to beginning my doctoral studies, I taught a second-year computer science course that had a set of pre-made slides compiled by past instructors. Here, I was exposed to the other side of the experience: if I just read off of slides, then what incentive do students have to come to class versus reading the slides from home? Could student interest in the course increase if they physically engaged with the material, say, through writing notes by hand or participating in live classroom activities?

While I opted during that course to deliver material using a combination of slides and chalkboard work, the question of slides and incentives never left my mind. As I began to hone my teaching skills, I observed various instructors and compared courses that were delivered primarily via slides to courses that were delivered primarily through other means, such as chalkboard work or seminar-style lectures. While I was aware of studies investigating the effects of technology in the university classroom, I had never heard of any study investigating technology specifically in a theoretical computer science course. Can the type of abstract material covered in such a course be delivered effectively using technology, or (like mathematics) are students better off learning such material "by hand", through writing notes and working offline? With more and newer classroom technology being introduced by the month, and especially with courses moving online in response to this year's pandemic, can—or should—courses in theoretical computer science change accordingly, and how will students be impacted?

Based on the existing literature, it seems that prior work can be distilled into three general statements:

1. PowerPoint slides should be used to communicate difficult or complex topics, especially if they can be presented in a visual manner;
2. PowerPoint slides do not have a noticeable effect on visual information retention, but may affect other forms of information retention; and
3. Students in engineering disciplines appreciate the use of PowerPoint slides for purposes such as note-taking, but do not have a strong preference for lectures that use PowerPoint slides.

However, I found it notable that I could not locate any research on the use of PowerPoint specifically in computer science lectures. Since computer science is such a vast discipline, it may be the case that some topics of study lend themselves more readily to PowerPoint-based lectures or other educational uses of technology. Thus, it is necessary for me to restrict the scope of my research question to the specific subdiscipline of theoretical computer science (and perhaps even to *introductory* theory courses, to ensure all students have the same baseline knowledge). Thus, I propose to investigate the following question:

"Does the use of presentation technology (e.g., PowerPoint slides) have a positive effect on student understanding in an undergraduate theoretical computer science course, compared to the use of a traditional (e.g., chalkboard) lecture style?"

Annotated Bibliography

1. Y. Inoue-Smith. College-based case studies in using PowerPoint effectively. *Cogent Education*, 3:1-15, 2016.

This paper examines the potential of PowerPoint as a tool to enhance traditional pedagogical techniques. Namely, the pedagogical technique under consideration in this paper is active learning, or the practice of engaging students during a lecture and allowing students to become involved with their own education. The paper takes a case study approach by considering the experiences of seven faculty members teaching across a variety of departments at an American university. All of the professors taught courses which are typically considered to belong to the general area of "liberal arts".

The paper focuses on a comparison between use of PowerPoint slides in a university lecture and students' perceptions of PowerPoint slide use. The author observes that three of the seven lectures were taught in a large lecture hall, where it is difficult to engage with students using a traditional lecture style (i.e., no PowerPoint). The author posits that using PowerPoint in a large lecture can both grab students' attention and help to structure lecture content, but slides should be written with the student in mind; teacher-centered slides do not give students an idea of how to approach the material on their own. Participants in the study claimed that using PowerPoint helps with lecture organization, but the responsibility for student learning remains with the professor. Before a professor creates slides for a course, they must take care to set specific objectives to measure a student's mastery of the material in the course.

The author specifically highlights in the conclusion of the paper that "Teaching mathematics with PowerPoint is not common; instead, most professors still use chalkboards. Chalk allows the mathematics professor to unveil each equation or theorem step by step." The author goes on to write that "PowerPoint is good for visually enriching the content and illustrating complex concepts", but it is easy to fall into the trap of overloading students with information or rushing too quickly through a presentation. Given that many theoretical computer science courses involve the use and construction of diagrams (e.g., for drawing finite automata), PowerPoint may be a good tool to enhance the visual aspect of a lecture.

2. C. Swati, T. Suresh, D. Sachin. Student assessment on learning based on PowerPoint versus chalkboard. *International Journal of Recent Trends in Science and Technology*, 13:347-351, 2014.

This paper investigates the impact of PowerPoint slide use versus chalkboard use on student learning, and the lecture delivery preferences of students in a microbiology department. It is a cross-sectional, interventional study: data was collected over a period of one term, and two groups of students received two lectures on topics in microbiology. The first topic was considered to be "simple", while the second topic was considered to be "complex". The content of each lecture was the same across both groups, and the instructors of each group had the same teaching experience. For the first lecture, one group was taught via chalkboard while the other was taught via PowerPoint slides, and for the second lecture, the teaching methods for each group were swapped. Students in both groups completed a pre-test before each lecture and a post-test following each lecture to evaluate their understanding of the material, with both tests containing the same set of questions.

The authors performed a statistical analysis on the pre-test and post-test performance data of both groups (hereafter referred to as "Group A" and "Group B", as in the paper). For the first lecture, Group A was taught via PowerPoint while Group B was taught via chalkboard. The pre-test mean score difference between Group A and Group B was not statistically significant, as was the difference in mean score change from pre-test to post-test between Group A and Group B. The proportion of passing students on the post-test versus on the pre-test was significant for both groups. For the second lecture, the pre-test mean score difference was again not significant, but the difference in mean score change from pre-test to post-test between both groups was significant. The proportion of passing students on the post-test versus on the pre-test was more significant for Group A (chalkboard) than for Group B (PowerPoint slides).

The authors conclude that traditional lecture delivery is more effective than PowerPoint slide lecture delivery in terms of raw scoring and student pass results. The authors note that chalkboard use resulted in a stronger focus and better lecture structure than PowerPoint slide use for complex topics. Since many computer science students consider theory to be a “complex topic”, this study suggests that it may be a subject better suited for traditional lecture delivery.

3. A. Savoy, R. W. Proctor, G. Salvendy. Information retention from PowerPoint and traditional lectures. *Computers and Education*, 52: 858-867, 2009.

This paper investigates the effects of PowerPoint slide use in lecture on three aspects of student outcomes on examinations: auditory scores, graphic scores, and alphanumeric scores. The paper studies students in a course that is cross-listed between departments of psychology and industrial engineering. The course was designed for engineering majors with little background in psychology. The study ran over a period of one term, and consisted of two lectures on differing topics. Both lectures were presented using a combination of traditional delivery and PowerPoint slides. Students were assessed on lecture content using a quiz consisting of multiple-choice questions; questions about lecture content were divided into questions about information presented auditorily by the instructor, information presented auditorily with visual support, information presented graphically, and alphanumeric information (i.e., text).

The authors conjectured that auditory information is difficult to recall when presented in conjunction with PowerPoint slides, and their findings showed that auditory scores with the traditional delivery style were 15% higher than with the PowerPoint delivery style. In terms of graphic scores, the authors observed no notable gain when using PowerPoint slides versus traditional lecturing, but the scores of students who attended class were higher than the scores for students who did not attend class, suggesting that attending class offers a tangible benefit over simply reading PowerPoint slides or texts from home. Combining both audio and visual content, the authors found no significant difference in scores between PowerPoint lectures and traditional lectures, suggesting that there is no benefit to using PowerPoint when lectures consist mostly of text and simple graphics. The authors finally conjectured that students would prefer PowerPoint slides over traditional lectures; subjective data collected by the authors showed that students had no strong preference for either lecture style.

This paper relates to Reference 2 in that both study a cross-section of students in a real-world lecture environment. However, given that the students in this study are engineering majors, their experience and personal inclinations may align more closely to the typical computer science student (i.e., they may be more technology-savvy). However, the fact that students in this study seemed not to have a strong opinion on PowerPoint lectures might suggest that technological ability does not translate to pedagogical preference.

4. A. O'Dwyer. Responses of engineering students to lectures using PowerPoint. In *Proceedings of the International Symposium for Engineering Education (ISEE 2008)*, pages 219-226, Dublin, 2008.

This paper, much like Reference 3, investigates the responses of engineering students to PowerPoint-based lectures versus traditional lectures. The authors survey three cohorts of engineering students across three different levels of study at one university. Between 65% and 75% of lecture time during the term involved use of PowerPoint across all three cohorts. Students were given a questionnaire at the end of the term where they were asked to compare lectures delivered using PowerPoint to lectures delivered in the traditional style (i.e., using chalkboards or overhead projector slides).

The authors found that students across all cohorts identified the value of using PowerPoint slides, and indicated that PowerPoint lectures were both more interesting and facilitated greater learning. Students also indicated a preference for having lecture material online in the form of PowerPoint slides, as they can add notes to printed copies of slides as a form of active learning. Lastly, students claimed to have more motivation to attend PowerPoint-based lectures, but claimed to feel less bad about missing a PowerPoint-based lecture over a traditional lecture.